SERIAL COMMUNICATION PROTOCOLS

In serial communication the data"s are transmitted bit-by-bit. The main advantage of serial communication over parallel communication is serial communication can be used for longer distance data transmission.

Data transmission Types:

1. Simplex
2. Half Duplex
3. Full Duplex

1. Simplex:
Data transfer takes place in only one direction.
2. Half Duplex:
Data transfer takes place in two directions, but not simultaneously.
3. Full duplex:
Data transfer takes place in two directions simultaneously.

I^2C (Inter Integrated Circuits) Protocols:

The **I^2C bus** is a well-known bus commonly used to link microcontrollers into systems. I_2C is designed to be low cost, easy to implement, and of moderate speed (up to 100 kilobits per second for the standard bus and up to 400 kbits/sec for the extended bus).

It uses only two lines: the **serial data line (SDL)** for data and the **serial clock line (SCL)**, which indicates when valid data are on the data line. Figure 1 shows the structure of a typical I_2C bus system. Every node in the network is connected to both SCL and SDL. Some nodes may be able to act as bus masters and the bus may have more than one master. Other nodes may act as slaves that only respond to requests from masters.

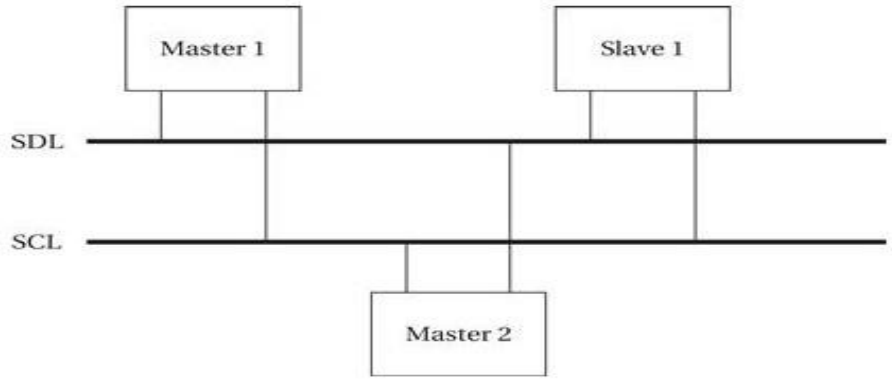


FIGURE 1 STRUCTURE OF AN I_2C BUS SYSTEM

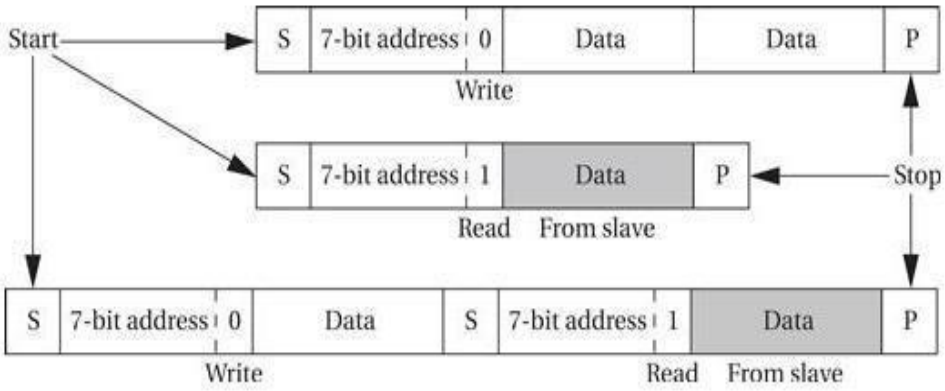
The I_2C bus is designed as a multimaster bus—any one of several different devices may act as the master at various times. As a result, there is no global master to generate the clock signal on SCL. Instead, a master drives both SCL and SDL when it is sending data. When the bus is idle, both SCL

and SDL remain high. When two devices try to drive either SCL or SDL to different values each master device must listen to the bus while transmitting to be sure that it is not interfering with another message.

Every I_2C device has an address. The addresses of the devices are determined by the system designer, usually as part of the program for the I_2C driver. The addresses must of course be chosen so that no two devices in the system have the same address. A device address is 7 bits in the standard I_2C definition (the extended I_2C allows 10-bit addresses). When a master wants to write a slave, it transmits the slave's address followed by the data. Because a slave cannot initiate a transfer, the master must send a read request with the slave's address and let the slave transmit the data. Therefore, an address transmission includes the 7-bit address and 1 bit for data direction: 0 for writing from the master tothe slave and 1 for reading from the slave to the master. The format of an address transmission is shown in Figure.

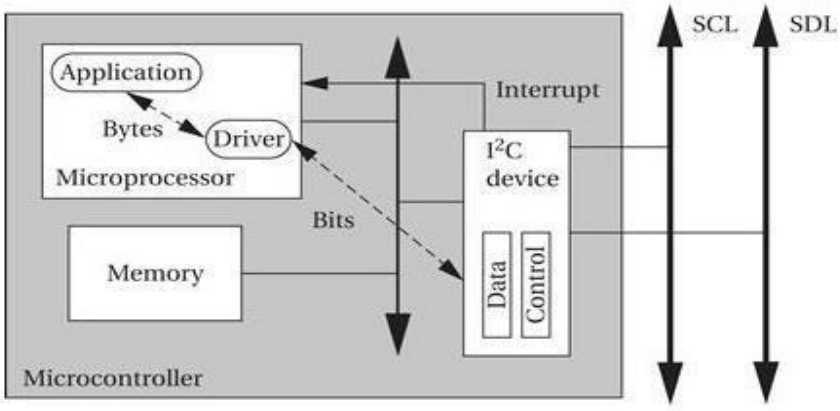
A bus transaction is initiated by a start signal and completed with an end signal:

❖ A start is signaled by leaving the SCL high and sending a 1 to 0 transition on SDL. ❖ A stop is signaled by setting the SCL high and sending a 0 to 1 transition on SDL.



The I_2C interface on a microcontroller can be implemented with varying percentages of the functional ity in software and hardware. As illustrated in Figure 2, a typical system has a 1bit hardware interface with routines for byte-level functions. The I_2C device takes care of generating the clock and data. The application code calls routines to send an address, send a data byte, and so on, which then generates the SCL and SDL, acknowledges, and so forth. One of the microcontroller's timers is typically used to control the length of bits on the bus. Interrupts may be used to recognize bits. However, when used in master mode, polled I/O may be acceptable if no other pending tasks can be performed, because masters initiate their own transfers.

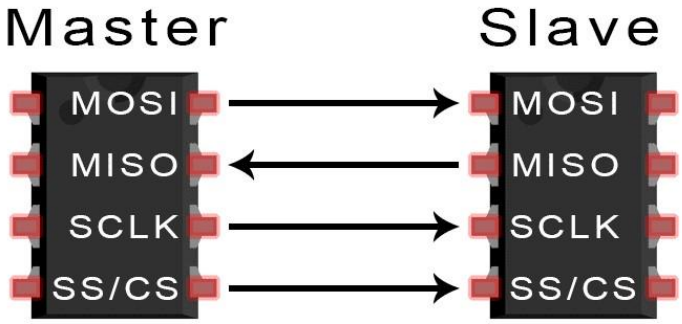
FIGURE 2



SPI Protocols:

SPI is a common communication protocol used by many different devices. For example, SD card reader modules, RFID card reader modules, and 2.4 GHz wireless transmitter/receivers all use SPI to communicate with microcontrollers. One unique benefit of SPI is the fact that data can be transferred without interruption. Any number of bits can be sent or received in a continuous stream. With I2C and UART, data is sent in packets, limited to a specific number of bits. Start and stop conditions define the beginning and end of each packet, so the data is interrupted during transmission.

Devices communicating via SPI are in a master-slave relationship. The master is the controlling device (usually a microcontroller), while the slave (usually a sensor, display, or memory chip) takes instruction from the master. The simplest configuration of SPI is a single master, single slave system, but one master can control more than one slave



MOSI (Master Output/Slave Input) – Line for the master to send data to the slave.

MISO (Master Input/Slave Output) – Line for the slave to send data to the master.

SCLK (Clock) – Line for the clock signal.

SS/CS (Slave Select/Chip Select) – Line for the master to select which slave to send data to.

How SPI Works

The clock signal synchronizes the output of data bits from the master to the sampling of bits by the slave. One bit of data is transferred in each clock cycle, so the speed of data transfer is determined by the frequency of the clock signal. SPI communication is always initiated by the master since the master configures and generates the clock signal.

Any communication protocol where devices share a clock signal is known as *synchronous*. SPI is a synchronous communication protocol. There are also *asynchronous* methods that don't use a clock signal. For example, in UART communication, both sides are set to a pre-configured baud rate that dictates the speed and timing of data transmission.

The clock signal in SPI can be modified using the properties of *clock polarity* and *clock phase*. These two properties work together to define when the bits are output and when they are sampled. Clock polarity can be set by the master to allow for bits to be output and sampled on either the rising or falling edge of the clock cycle. Clock phase can be set for output and sampling to occur

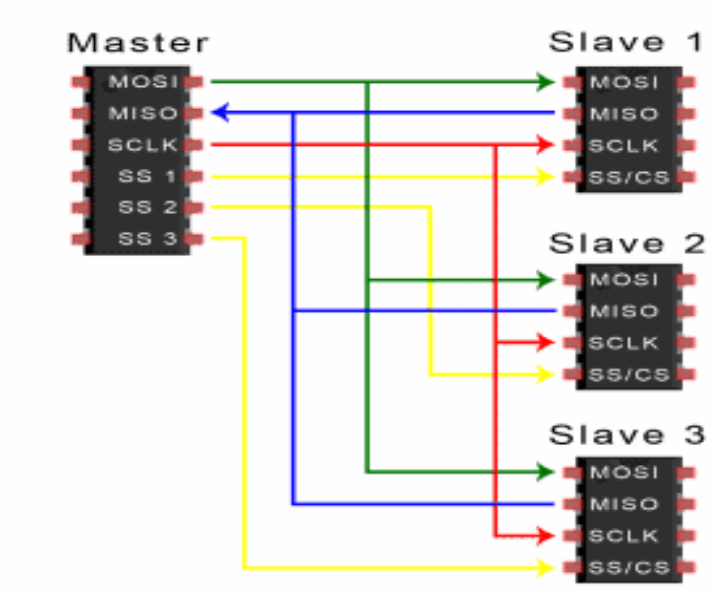
on either the first edge or second edge of the clock cycle, regardless of whether it is rising or falling.

Slave Select

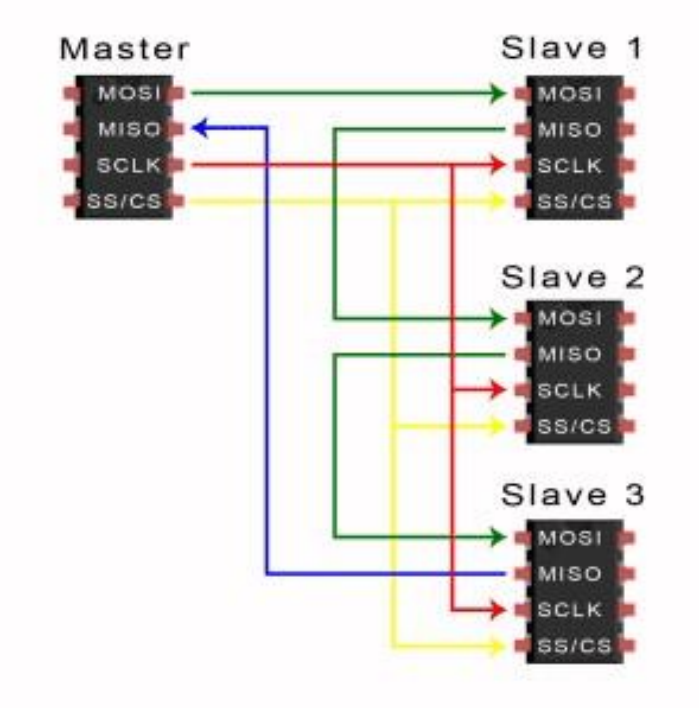
The master can choose which slave it wants to talk to by setting the slave's CS/SS line to a low voltage level. In the idle, non-transmitting state, the slave select line is kept at a high voltage level. Multiple CS/SS pins may be available on the master, which allows for multiple slaves to be wired in parallel. If only one CS/SS pin is present, multiple slaves can be wired to the master by daisy-chaining.

Multiple Slaves

SPI can be set up to operate with a single master and a single slave, and it can be set up with multiple slaves controlled by a single master. There are two ways to connect multiple slaves to the master. If the master has multiple slave select pins, the slaves can be wired in parallel like this:



If only one slave select pin is available, the slaves can be daisy-chained like this:



MOSI and MISO

The master sends data to the slave bit by bit, in serial through the MOSI line. The slave receives the data sent from the master at the MOSI pin. Data sent from the master to the slave is usually sent with the most significant bit first.

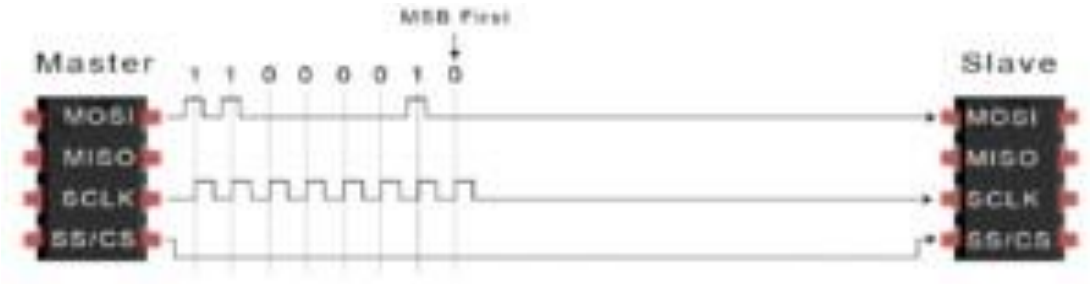
The slave can also send data back to the master through the MISO line in serial. The data sent from the slave back to the master is usually sent with the least significant bit first.

Steps of SPI Data Transmission

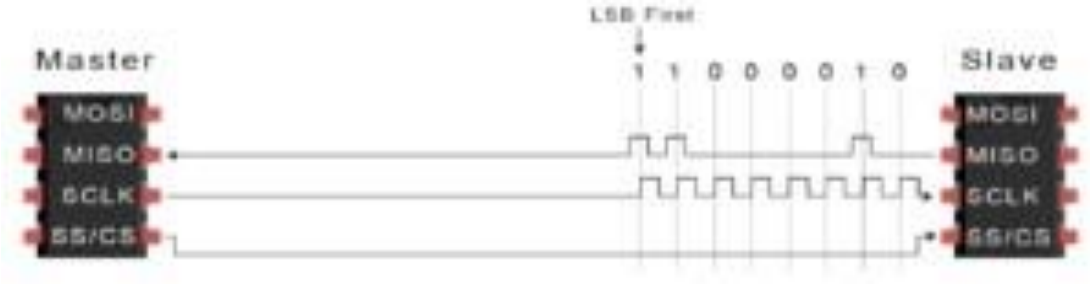
1. The master outputs the clock signal:

2. The master switches the SS/CS pin to a low voltage state, which activates the slave:

3. The master sends the data one bit at a time to the slave along the MOSI line. The slave reads the bits as they are received:



4. If a response is needed, the slave returns data one bit at a time to the master along the MISO line. The master reads the bits as they are received:



Advantages and Disadvantages of SPI

Advantages

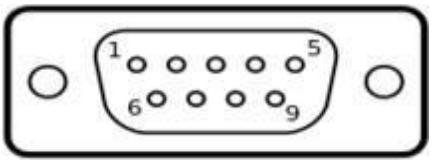
- No start and stop bits, so the data can be streamed continuously without interruption
- No complicated slave addressing system like I2C
- Higher data transfer rate than I2C (almost twice as fast)
- Separate MISO and MOSI lines, so data can be sent and received at the same time

Disadvantages

- Uses four wires (I2C and UARTs use two)
- No acknowledgement that the data has been successfully received (I2C has this)
- No form of error checking like the parity bit in UART
- Only allows for a single master

RS232C "Recommended Standard 232C":

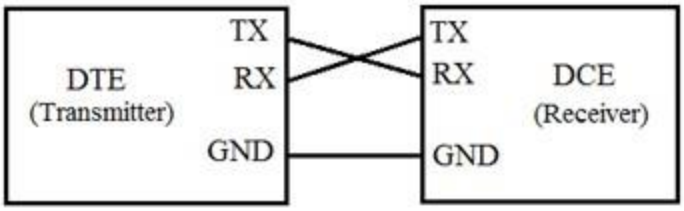
It was a widely-used version featuring 25-pin connectors, while ***RS232D*** introduced 22-pin configurations. Modern implementations typically use 9-pin D-type male connectors (DB9) which include the most essential signals for practical communication. The current ***TIA-232-F standard*** maintains compatibility with all previous connector configurations while



providing enhanced electrical specifications.

Fig: DB – 9, 9- PIN CONNECTOR

- RS232 is a standard protocol used for serial communication, it is used for connecting computer and its peripheral devices to allow serial data exchange between them. As it obtains the voltage for the path used for the data exchange between the devices. It is used in serial communication up to 50 feet with the rate of 1.492kbps. As EIA defines, the RS232 is used for connecting **Data Transmission Equipment (DTE)** and **Data Communication Equipment (DCE)**.



How RS232 Works?

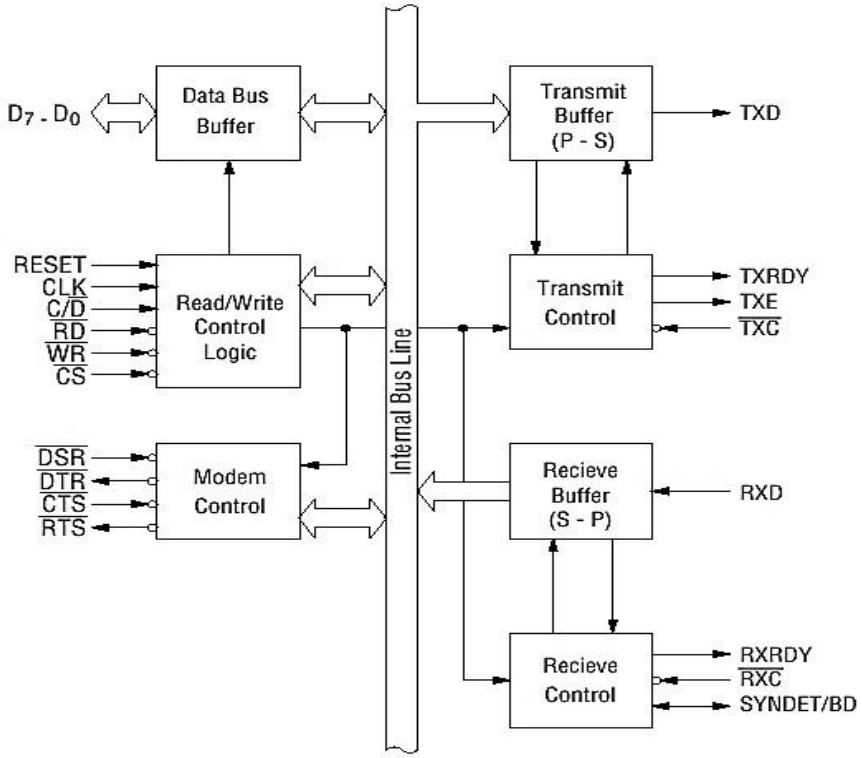
RS232 works on the two-way communication that exchanges data to one another. There are two devices connected to each other, **(DTE) Data Transmission Equipment& (DCE) Data Communication Equipment** which has the pins like **TXD, RXD, and RTS& CTS**. Now, from **DTE** source, the **RTS** generates the *request to send* the data. Then from the other side **DCE**, the **CTS,** clears the path for receiving the data. After clearing a path, it will give a signal to **RTS** of the **DTE** source to send the signal. Then the bits are transmitted from **DTE** to **DCE**. Now again from **DCE** source, the request can be generated by **RTS** and **CTS** of **DTE** sources clears the path for receiving the data and gives a signal to

send the data. This is the whole process through which data transmission takes place.

RS232 is the most widely used serial I/O interfacing standard. This standard is used in PCs and numerous types of equipment. In RS232, 1 is represented as 3 to -15 V and 0 as +3 to +15 V. To connect RS232 to microcontroller system we use voltage converters such as MAX232 and MAX233. The RS232 standard is not TTL compatible; therefore, it requires a line driver such as MAX232 chip to convert RS232 voltage levels to TTL levels, and vice versa.

USART

8251 is a universal synchronous asynchronous receiver and transmitter. This chip converts the parallel data into a serial stream of bits suitable for serial transmission. It also receives serial stream of data bits and convert it into parallel data bytes.



Data bus buffer:
This is a bidirectional buffer that interfaces internal circuit with system bus.

Read/write control logic:
It controls the operation of the peripheral depending upon the operation initiated by the CPU.

Modem control:
It handles the handshake signals to coordinate the communication between modem and USART.
- **Data Set Ready**
It is used to check if the data set is ready when communicating with the modem.

- Data Terminal Ready

It indicates that the device is ready to accept data when 8251 is communicating with a modem. -

Clear To Send

When = 0, it clears all the data present in the modem to allow further communication.

- Request To Send

It indicates that the receiver is ready to receive a data byte from the modem.

Transmit buffer:

The transmit buffer is a parallel to serial converter that receives a parallel byte and converts it into a serial byte.

TXD - Transmit Data

It transmits data bits along with other information like start bit, stop bit and parity bit.

Transmit control:

It transmits data bits received by the data buffer from CPU for further serial communication.

TXRDY - Transmitter Ready

It indicates that 8251 is ready to receive a character.

TXE - Transmitter empty

It indicates that 8251 has no character to transmit while transmitting

- Transmitter Clock

It controls the rate at which the character is to be transmitted.

Receive Buffer:

It receives the serial data.

RXD - Receive Data

Receives a stream of data to be received by 8251.

Receive control:

RXRDY - Receiver Ready

Indicates that 8251 has a character in the buffer register and it is ready to transfer it to CPU.

- Receiver Clock

It controls the rate at which the character is to be received.

SYNDET/BD – Synch Detect/ Break Detect In synchronous mode this in is used for detecting SYNC characters and may be used as either input or output.

In asynchronous mode this pin acts as a break detect output.